A Framework for Effective Placement of Virtual Machine Replicas for Highly Available Performance-sensitive Cloud-based Applications

Kyoungho An, Faruk Caglar, Shashank Shekhar, Aniruddha Gokhale
Department of Electrical Engineering and Computer Science
Vanderbilt University
Nashville, TN 37235, USA
Email: {kyoungho.an, faruk.caglar, shashank.shekhar, a.gokhale}@vanderbilt.edu

***Abstract*—Applications are increasingly being deployed in the Cloud due to benefits stemming from economy of scale, scalability, flexibility and utility-based pricing model. Although most cloud-based applications have hitherto been enterprise-style, there is a new trend towards hosting performance-sensitive applications in the cloud that demand both high availability and good response times. In the current state-of-the-art in cloud computing research, there does not exist solutions that provide both high availability and acceptable response times to these applications in a way that also optimizes resource consumption in data centers, which is a key consideration for cloud providers. This paper addresses this dual challenge by presenting a design of a fault-tolerant framework for virtualized data centers that makes two important contributions. First, it describes an architecture of a fault-tolerance framework that can be used to automatically deploy replicas of virtual machines in data centers in a way that optimizes resources while assures availability and responsiveness. Second, it describes a specific formulation of a replica deployment combinatorial optimization problem that can be plugged into our strategizable deployment framework.**

***Keywords*-fault-tolerance, cloud computing, framework, availability and response times.**

I. Introduction

Cloud computing is a large-scale distributed computing paradigm based on the principles of utility computing that offers a variety of resources such as CPU and storage, systems software, and applications as services over the Internet [1]. The primary driving force behind the success of cloud computing is economy of scale. Traditionally although the cloud has been used to support enterprise applications, lately a class of performance-sensitive applications that demand both high availability and good response times are moving towards cloud-based hosting [2], [3].

Supporting performance-sensitive applications in the cloud requires that the cloud infrastructure be able to meet the response time, reliability and high availability requirements of these applications. A scrutiny of contemporary solutions for cloud computing reveals that the algorithms and mechanisms used to support applications in the cloud are tailored to meet the performance and reliability requirements of enterprise applications and not the performance-sensitive applications. To address the more stringent quality of service (QoS) requirements of these applications including good response times and high availability, new algorithms and techniques will need to be designed to manage the different cloud platform entities, such as the service architecture, data center network architecture, and virtualized resources.

Addressing the high availability and reliability concerns of performance-sensitive applications will require redundant virtual machines (VMs) and state synchronization among replica VMs as well as migrations of VMs. Traditional statically defined schemes of replication placement and resource allocation will often not be applicable due to the changing dynamics of a cloud environment. Thus, in this paper we focus on a fault tolerance architecture in the cloud geared to provide high availability and reliability for performance-sensitive applications. Moreover, to assure good response times to applications despite failures while also optimally utilizing resources, we present an Integer Linear Programming (ILP) formulation of a problem that allocates VMs and their replicas to physical resources in a data center that satisfies the QoS requirements of the applications. This and many other VM replica allocation algorithms can be designed and plugged into the framework we are building.

The rest of this paper is organized as follows: Section II describes relevant related work comparing it with our contributions; Section III describes the system architecture that provides high availability solutions to performance-sensitive applications hosted in the cloud; Section IV presents the ILP formulation; and Section V presents concluding remarks alluding to future work.

II. Related Work

This section presents related work and compares it with our contributions. In particular, we organize the related work along three dimensions as described below.

Autonomic Virtual Machine Placement: A related work closest to ours appears in [4]. The authors present a prototype of a VM placement system where an autonomic controller dynamically manages the mapping of VMs onto physical hosts according to policies specified by the user. As in our case, they too suggest a system architecture

for autonomic virtual machine placement in accordance with algorithms defined by users. However, our research additionally considers dynamic allocation of VM replicas with high-availability solutions to accomplish a fault-tolerant cloud computing architecture. Therefore, our case takes a different problem model.

Placement Algorithms: Lee et al. [5] investigate the VM consolidation heuristics to discover the assumptions on how virtual machines operate when the VMs reside in the same host machine. Additionally, they explore how the dimensions of resource information such as CPU, memory, and network bandwidth is in effect to augment the benefits of VM consolidation. The work in [6] proposes a modified Best Fit Decreasing (BFD) algorithm as VM reallocation heuristics for an efficient resource management. The evaluation in the paper shows that the suggested heuristics minimize energy consumption with providing high Quality of Service (QoS). While our work may benefit from these prior works, we are additionally concerned with placing replicas in a way that after failover the applications will continue to obtain the desired QoS.

High Availability Solutions: Our solution leverages the continuous check-pointing based Remus [7] high availability solution to achieve fault tolerance in cloud, which we have explained later. However, there are several other solutions available. VMware fault tolerance [8] is one of them which runs primary and backup VMs in lock-step [9] using deterministic replay. This keeps both the VMs in sync but it requires execution at both the VMs and needs network connection of high quality.

Kemari [10] is another approach which takes advantage of both lock-stepping and continuous check-pointing approaches. It synchronizes primary and secondary VMs just before the primary VM has to send an event to devices such as storage and networks. At this point, the primary VM pauses and Kemari updates the state of secondary VM to the current state of primary VM. Thus, VMs are synchronized with less complexity compared to lock-stepping and output latency of continuous check-pointing due to external buffering mechanism is also avoided.

Another important work on high availability is HydraVM [11]. It is storage based, memory efficient high availability solution which does not need a passive memory reservation for backups. It uses incremental check-pointing like Remus, but it maintains a complete recent image of VM in shared storage instead of memory replication. Thus, it reduces hardware costs for providing high availability support and provides greater flexibility as recovery can happen on any physical host having access to shared storage.

III. System Architecture for Delivering High Availability and Response Times

This section presents our high availability system architecture to support performance-sensitive applications in the cloud data centers. We show how the architecture supports a pluggable VM replica allocation mechanism that can be used to utilize resources optimally while providing good response times to applications despite failures.

A. Overview of Remus and ACE

We briefly cover Remus and ACE, which are two software architectures we leverage in our work.

Remus [7] is a software system that provides OS- and application-agnostic high availability on commodity hardware. The choice of Remus is based on the fact that it provides seamless failure recovery and does not require lock step-based whole-system replication. The use of speculative execution [7] in the Remus approach ensures that the performance degradation due to replication is kept to a minimum. Speculative execution decouples the execution of the application from the synchronization issues. Since Remus provides protection against single host fail-stop failures only, if both the primary and backup hosts fail concurrently, the failure recovery will not be seamless; however, Remus ensures that the system's data will be left in a crash consistent state.

The ADAPTIVE Communication Environment (ACE) [12] is an open-source object-oriented framework for concurrent communication software. The ACE framework supports the communication software tasks including event demultiplexing and event handler dispatching, service initialization, inter-process communication, shared memory management, dynamic reconfiguration of distributed services, and concurrent execution and synchronization. For our work, we make use of the ACE Reactor framework [13] for dispatching events, the ACE Service Configurator framework for configuring software components, and the ACE Common Data Representation (CDR) for implementing the communication protocol between remotely located software components.

B. System Architecture

The conceptual system design of our proposed system is illustrated in Figure 1. The system of interest is the block with blue dashed line, and comprises Local Fault Manager (LFM) and Global Fault Manager (GFM) applications. The inputs of the system to these manager entities are resource information of physical hosts and VMs gathered directly from a virtual machine hypervisor. CPU, memory, network, storage, and processes are some of the resource information used in our system. The LFM and GFM applications are responsible for deploying VM replicas in data centers.

The LFM retrieves the resource information from a VM, and passes that information periodically to the GFM. The GFM employs a pluggable deployment algorithm framework and a replication manager to decide where the replica of a VM should reside. Replication manager is the core component of the GFM and is responsible to run the deployment

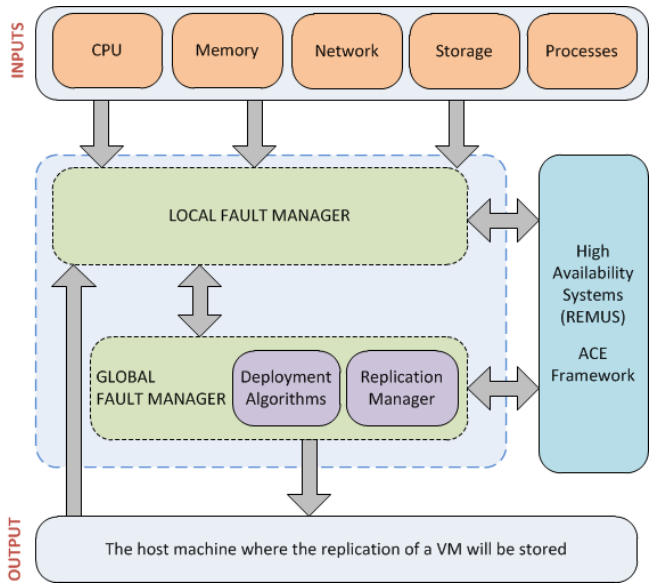

Figure 1. Conceptual System Design

algorithm plugged in by the user in the framework. The output of the system is the mapped host machine where the replica of a VM will be stored. This output is then supplied to the LFM running on the host machine where the VM is located to take the necessary actions. Both the LFM and GFM interact with existing high availability services (*e.g.*, Remus) provided in hypervisors like Xen, and communication middleware frameworks, such as ACE.

The LFM runs a High Availability Service (HAS) to serve the high availability functionality for VMs under its purview. This solution is based on the Strategy pattern [14] wherein the HAS provides an interface for managing the high availability functionality. This includes starting and stopping the replication operations, and automatic failover from primary VM to the backup VM in case of a failure.

Once the HAS is started and while it is operational, it keeps replicating state from primary VM to backup VM. If a failure occurs during this period, it switches to the backup VM making it the active VM. When HAS is stopped, it stops the replication process and high availability is discontinued. This functionality can be provided using any suitable high availability solution in the cloud environment using the Strategy pattern, which enables us the flexibility to change the high availability solution if and when needed.

In the context of HAS, the job of the GFM is to provide LFM with backup VMs which can be utilized by HAS for providing high availability. In the event of failure of primary VM, HAS ensures that the processing switches to the backup VM and it becomes the primary VM. Simultaneously, the LFM informs GFM of the failure event and requests additional backup VMs on which a replica can start. It is the GFM's responsibility to provide resource to LFM in a timely manner so that it can move from crash consistent state to seamless recovery fault tolerant state as soon as possible thereby assuring average response times of performance-sensitive applications.

C. Virtual Machine Replica Placement

Deployment algorithms are used to determine which host machine should store the VM and VM replica in the context of fault management. There are different types of algorithms to make this decision. Optimization algorithms, such as bin packing, genetic algorithms, multiple knapsack, and simulated annealing are some of the choices used in a large number of industrial applications today. Moreover, different heuristics of the bin packing algorithm are commonly utilized techniques for VM replica placement optimization, in particular.

In bin packing algorithms [15], the goal is to use minimum number of bins to pack the items of different sizes. Best-Fit, First-Fit, First-Fit-Decreasing, Worst-Fit, Next-Fit, and Next-Fit-Decreasing are the different heuristics of this algorithm. All these heuristics will be part of the middleware we are designing, and will be provided to the framework user to run the bin packing algorithm.

In our project, we identify VMs and host machines as the item and bin elements of the algorithm, respectively. Resource information of a VM, such as CPU, memory, and network are utilized as their weights to employ the bin packing algorithm. Resource information are aggregated into a one single scalar value, and one dimensional bin packing is employed to find the best host machine where the replication of a VM will be stored. Our framework allows plugging in of different VM replica placement algorithms. A concrete technique we have used in our replication manager is described in Section IV.

D. Providing High Availability

In the architecture shown in the Figure 2, replicas of VMs are automatically deployed in hosts assigned by a GFM and LFMs. The following are the steps of the system described in the figure.

1) A GFM service is started, and the service waits for connections from LFMs.
2) LFMs will join the system by connecting to the GFM service.
3) The joined LFMs send resource information of VMs and physical hosts such as CPU, memory, and network bandwidth to the GFM.
4) Based on the resource information, the GFM determines an optimal deployment plan for joined physical hosts and VMs by running a deployment algorithm which can be selected by users.
5) The GFM will notify LFMs to execute high availability processes in LFMs with information of source VMs and destination hosts.

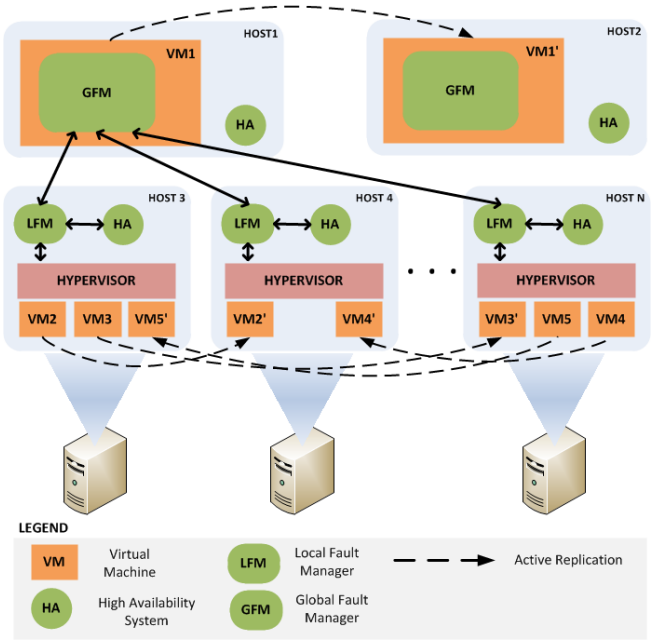

Figure 2. System Architecture

A GFM service can be deployed on a physical machine host or a virtual machine host. In our system design, to accomplish the high availability of a GFM service, a GFM is deployed in a VM and a GFM's VM replica is located in another physical host. When the host where the GFM is located fails, the backup VM containing the GFM service is up and running to continue the GFM service via a high availability process.

The GFM framework provides an interface for a deployment algorithm. Through the framework interfaces, system developers can implement their own algorithms and examine performance of the algorithms comparing to other existing algorithms. The Strategy pattern is used to provide a more flexible way for system developers to implement the algorithms and change the deployment algorithm at run-time for system administrators. The ACE Reactor framework [13] is used to detect and dispatch the events occurred in the GFM, and the ACE CDR is used to process a stream of data from LFMs by a defined protocol between a GFM and LFMs.

LFMs are placed in physical hosts used to run VMs in data centers. LFMs work with a hypervisor and a high availability system to collect resource information of VMs and hosts and to replicate VMs to other backup hosts, respectively.Through the high availability solution, a VM's disk, memory, and network connections are actively replicated to other hosts and a replication of the VM in a backup host is instantiated when the primary VM is failed. Like a GFM, to implement the LFM framework, the ACE Reactor framework is used to detect and dispatch events such as time out events for sending resource information and input events from a GFM to control processes of high availability systems. The ACE CDR is also used to process a stream of data from a GFM according to the protocol. A LFM connects and gets resource information from a hypervisor via the libvirt library.

E. System Configuration

Figure 3 shows the use case diagram of the system in which roles and responsibilities of software components are defined. A user as a system administrator should configure and run a GFM service and LFM services. A user as a system developer can implement deployment algorithms to find and use a better deployment solution. The LFM services continuously update resource information of VMs and hosts using a configured interval by a user. The GFM service uses the deployment algorithms and the resource information to create a deployment plan for replications of VMs. Then, the GFM sends messages to LFMs to run a backup process via high availability solutions.

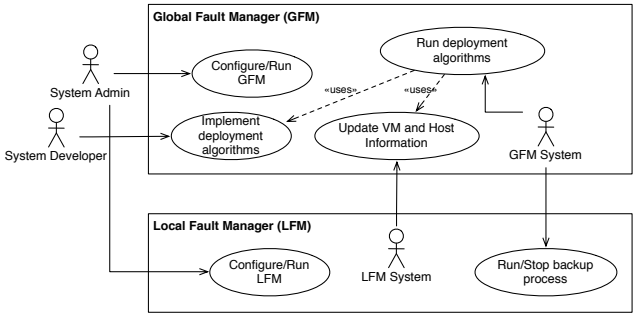

Figure 3. Use Case Diagram

IV. Integer Linear Programming Problem Formulation

One of our goals is to develop a framework that enables plugging in different user-supplied placement algorithms. We expect that our framework will compute replica placement decisions in an online manner in contrast to making offline decisions. This section describes an instance of VM replica placement algorithm we have developed. We have formulated it as an Integer Linear Programming (ILP) problem.

In our ILP formulation we assume that a data center comprises multiple hosts. Each host can in turn comprise multiple VMs. We also account for the utilizations of the physical host as well as the VMs on each host. Furthermore, not only do we account for CPU utilizations but also memory and network bandwidth usage. All of these resources must be accounted for in determining the placement of the replicas because on a failover we expect our applications to continue to receive their desired QoS properties. Table I describes the variables used in our ILP formulation.

Table I
NOTATION AND DEFINITION OF THE ILP FORMULATION

Notation	Definition
x_{ij}	Boolean value to determine the i^{th} VM to the j^{th} physical host mapping
x'_{ij}	Boolean value to determine the replication of the i^{th} VM to the j^{th} physical host mapping
y_j	Boolean value to determine usage of the physical host j
c_i	CPU usage of the i^{th} VM
c'_i	CPU usage of the i^{th} VM's replica
m_i	Memory usage of the i^{th} VM
m'_i	Memory usage of the i^{th} VM's replica
b_i	Network bandwidth usage of the i^{th} VM
b'_i	Network bandwidth usage of the i^{th} VM's replica
C_j	CPU capacity of the j^{th} physical host
M_j	Memory capacity of the j^{th} physical host
B_j	Network bandwidth of the j^{th} physical host

We now present the ILP problem formulation with defined constraints that need to be satisfied to find an optimal allocation of VM replicas.

$$\text{minimize} \quad \sum_{j=1}^{m} y_i \tag{1}$$

$$\text{subject to} \quad \sum_{j=1}^{m} x_{ij} = 1 \qquad \forall i \tag{2}$$

$$\sum_{j=1}^{m} x'_{ij} = 1 \qquad \forall i \tag{3}$$

$$\sum_{i=1}^{n} c_i x_{ij} + \sum_{i=1}^{n} c'_i x'_{ij} \leq C_j y_j \qquad \forall j \tag{4}$$

$$\sum_{i=1}^{n} m_i x_{ij} + \sum_{i=1}^{n} m'_i x'_{ij} \leq M_j y_j \qquad \forall j \tag{5}$$

$$\sum_{i=1}^{n} b_i x_{ij} + \sum_{i=1}^{n} b'_i x'_{ij} \leq B_j y_j \qquad \forall j \tag{6}$$

$$\sum_{i=1}^{n} x_{ij} + \sum_{i=1}^{n} x'_{ij} = 1 \qquad \forall j \tag{7}$$

$$x_{ij} = \{0,1\},\ x'_{ij} = \{0,1\},\ y_j = \{0,1\} \tag{8}$$

The objective function of the problem is to minimize the number of physical hosts by satisfying the requested resource requirements of VMs and their replicas. Constraints (2) and (3) ensure every VM and VM's replica is deployed in a physical host. Constraints (4), (5), (6) guarantee a total capacity of CPU, memory, and network bandwidth of deployed VMs and VMs' replicas are packed into an assigned physical host, respectively. Constraint (7) checks that a VM and its replica is not deployed in the same physical host since the physical host may become a single point of failure, which must be prevented.

V. CONCLUSION

As performance-sensitive applications move to the cloud, it becomes important for cloud platforms to implement algorithms that provide the QoS properties (*e.g.*, timeliness, high availability, reliability) of these applications. In turn this implies providing algorithms and mechanisms for effective fault tolerance and assuring application response times while simultaneously utilizing resources optimally. Thus, the desired solutions require a combination of algorithms for managing and deploying replicas of virtual machines on which the performance-sensitive applications are deployed in a way that optimally utilizes resources, and algorithms that ensure timeliness and high availability requirements.

This paper presented our preliminary work in this area describing a framework we are developing. The paper presented the architectural details of a framework for a fault-tolerant cloud computing infrastructure that can automatically deploy replicas of VMs according to flexible algorithms defined by users. Finding an optimal placement of VM replicas in data centers is an important problem to be resolved because it determines the QoS delivered to performance-sensitive applications running in the cloud. To that end this paper presents an instance of an online VM replica placement algorithm we have formulated as an ILP problem.

The work presented in this paper is not sufficient to address the vast number of challenges. For example, scheduling of virtual machines (VMs) on the host operating system (OS) and in turn scheduling of applications on the guest OS of the VM in a way that assures application response times is a key challenge that needs to be resolved. Scheduling alone is not sufficient; the resource allocation problem must be addressed wherein physical resources including CPU, memory, disk and network must be allocated to the VMs in a way that will ensure that application QoS properties are satisfied. In doing so, traditional solutions used for hard real-time systems based on over-provisioning are not feasible because the cloud is an inherently shared infrastructure, and operates on the utility computing model. Autoscaling algorithms used in current cloud computing platforms must be such that response times are not adversely impacted when resources are scaled up or down, and applications must be migrated.

The gamut of the problem space described above is vast. Addressing these needs forms the bulk of our future work. Our ongoing research is focusing on implementing the

proposed architecture, and providing a framework that enables pluggability of VM placement algorithms. Additional dimensions of future work involves validating our approach on a variety of performance-sensitive applications hosted in the cloud. To that end we are leveraging a private cloud testbed we have deployed at our institution where we have access to a variety of latest hardware and network switches, as well as a variety of open-source cloud infrastructure platforms, such as OpenStack and OpenNebula as well as hypervisors, such as Xen and KVM.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation NSF SHF/CNS Award CNS 0915976 and NSF CAREER CNS 0845789. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A View of Cloud Computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] A. Corradi, L. Foschini, J. Povedano-Molina, and J. Lopez-Soler, "DDS-enabled Cloud Management Support for Fast Task Offloading," in *IEEE Symposium on Computers and Communications (ISCC '12)*, Jul. 2012, pp. 67–74.

[3] T. M. Takai, "Cloud Computing Strategy," Department of Defense Office of the Chief Information Officer, Tech. Rep., Jul. 2012. [Online]. Available: http://www.defense.gov/news/DoDCloudComputingStrategy.pdf

[4] C. Hyser, B. McKee, R. Gardner, and B. Watson, "Autonomic virtual machine placement in the data center," *Hewlett Packard Laboratories, Tech. Rep. HPL-2007-189*, 2007.

[5] S. Lee, R. Panigrahy, V. Prabhakaran, V. Ramasubrahmanian, K. Talwar, L. Uyeda, and U. Wieder, "Validating heuristics for virtual machines consolidation," *Microsoft Research, MSR-TR-2011-9*, 2011.

[6] A. Beloglazov and R. Buyya, "Energy efficient allocation of virtual machines in cloud data centers," in *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*. Ieee, 2010, pp. 577–578.

[7] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield, "Remus: High Availability via Asynchronous Virtual Machine Replication," in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*. USENIX Association, 2008, pp. 161–174.

[8] D. J. Scales, M. Nelson, and G. Venkitachalam, "The design of a practical system for fault-tolerant virtual machines," *Operating Systems Review*, vol. 44, no. 4, pp. 30–39, 2010.

[9] T. C. Bressoud and F. B. Schneider, "Hypervisor-Based Fault Tolerance," *ACM Trans. Comput. Syst.*, vol. 14, no. 1, pp. 80–107, 1996.

[10] Y. Tamura, K. Sato, S. Kihara, and S. Moriai, "Kemari: Virtual machine synchronization for fault tolerance," *In USENIX 2008 Poster Session*, 2008.

[11] K.-Y. Hou, M. Uysal, A. Merchant, K. G. Shin, and S. Singhal, "Hydravm: Low-cost, transparent high availability for virtual machines," HP Laboratories, Tech. Rep., 2011.

[12] D. C. Schmidt, "The ADAPTIVE Communication Environment: An Object-Oriented Network Programming Toolkit for Developing Communication Software," in *Proceedings of the* 12^{th} *Annual Sun Users Group Conference*. San Jose, CA: SUG, Dec. 1993, pp. 214–225.

[13] ——, "Reactor: An Object Behavioral Pattern for Concurrent Event Demultiplexing and Event Handler Dispatching," in *Pattern Languages of Program Design 1*, J. O. Coplien and D. C. Schmidt, Eds. Reading, Massachusetts: Addison-Wesley, 1995, pp. 529–545.

[14] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1995.

[15] J. Berkey and P. Wang, "Two-dimensional finite bin-packing algorithms," *Journal of the operational research society*, pp. 423–429, 1987.